**HO CHI MINH CITY   
UNIVERSITY OF TECHNOLOGY AND EDUCATION**

**FACULTY OF HIGH QUALITY TRANING**

**🙡🙟🕮🙝🙣**

**COMPUTER ORGANIZATION AND ARCHITECTURE**

**PROGRAMMING THE 8051 MICROCONTROLLER USING THE C PROGRAMMING LANGUAGE INTERRUPTS AND COMMUNICATES**

|  |
| --- |
| Logo  Description automatically generated |

**Lecturer: Ph.D. Pham Van Khoa**

**Class: COOA335364E\_21\_2\_02CLC**

***Ho Chi Minh City, May 2022***

**LIST OF MEMBERS**

**SEMESTER 2 - ACADEMIC YEAR: 2021-2022**

# DISTRIBUTION TABLE

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Vũ Văn Đức** | **Nguyễn Tiến Đạt** | **Mai Xuân Hoài** | **Nguyễn Minh Mẫn** | **Nguyễn Hồng Đăng** |
| **Chapter 1** |  |  |  |  | **√** |
| **Chapter 2** |  |  | **√** | **√** |  |
| **Chapter 3** | **√** | **√** |  |  |  |
| **Conclusion** |  |  |  |  | **√** |
| **Simulation** | **√** | **√** | **√** |  |  |
| **Report** | **√** | **√** | **√** | **√** | **√** |
| **PowerPoints** | **√** | **√** | **√** | **√** | **√** |

# Table of Contents

[**DISTRIBUTION TABLE**](#_Toc106539474)

[**Table of Contents 1**](#_Toc106539475)

[**CHAPTER 1: INTRODUCTION 3**](#_Toc106539476)

[**1.1 Introduction to Computer organization and architecture. 3**](#_Toc106539477)

[**1.2 Introduction to 8051 microcontrollers. 3**](#_Toc106539478)

[**1.2.1 Brief History of 8051 3**](#_Toc106539479)

[**1.2.2 Comparison between 8051 Family Members 4**](#_Toc106539480)

[**1.2.3 Features of 8051 Microcontroller 5**](#_Toc106539481)

[**1.2.4 Block Diagram and architechure of 8051 Microcontroller 6**](#_Toc106539482)

[**1.2.5 8051 Microcontroller Pin diagram 7**](#_Toc106539483)

[**CHAPTER 2: PROGRAMMING INTERRUPT THE 8051 MICROCONTROLLERS USING THE C PROGRAMMING LANGUAGE. 9**](#_Toc106539484)

[**2.1 Interrupt in 8051 9**](#_Toc106539485)

[**2.2 Types of interrupts in 8051 9**](#_Toc106539486)

[**2.3 Interrupt priorities 10**](#_Toc106539487)

[**2.4 The process when performing an interrupt 10**](#_Toc106539488)

[**2.5 Interrupt stucture of 8051 Microcontroller 11**](#_Toc106539489)

[**2.6 Interrupt Timer 13**](#_Toc106539490)

[**2.6.1 Block diagram 13**](#_Toc106539491)

[**2.6.2 Application 14**](#_Toc106539492)

[**2.6.3 Programming interrupt in C 14**](#_Toc106539493)

[**2.6.4 Circuit diagram. 15**](#_Toc106539494)

[**2.6.5 Simulation 15**](#_Toc106539495)

[**2.7 Interrupt external 16**](#_Toc106539496)

[**2.7.2 Application 17**](#_Toc106539497)

[**2.7.3 Programming interrupt in C 17**](#_Toc106539498)

[**2.7.4 Circuit diagram. 18**](#_Toc106539499)

[**2.7.5 Simulation 18**](#_Toc106539500)

[**CHAPTER 3: PROGRAMMING UART THE 8051 MICROCONTROLLERS USING THE C PROGRAMMING LANGUAGE. 20**](#_Toc106539501)

[**3.1 Introduction to UART 20**](#_Toc106539502)

[**3.1.1 Registers used for UART 20**](#_Toc106539503)

[**3.1.2 Baud rate in the 8051 22**](#_Toc106539504)

[**3.2 Serial – Parallel Transfer 23**](#_Toc106539505)

[**3.3 Types of Data Transmission 24**](#_Toc106539506)

[**a) Simplex mode 24**](#_Toc106539507)

[**b) Half-duplex mode 24**](#_Toc106539508)

[**c) Full-duplex mode 24**](#_Toc106539509)

[**d) Transmission mode in 8051 25**](#_Toc106539510)

[**3.4 Method of serial communication 27**](#_Toc106539511)

[**3.5 Interface standard. 28**](#_Toc106539512)

[**a) RS 232 28**](#_Toc106539513)

[**b) Max232 29**](#_Toc106539514)

[**3.6 UART – Universal Asynchronous Receiver/Transmitter 30**](#_Toc106539515)

[**3.6.1 Block diagram 30**](#_Toc106539516)

[**3.6.2 Programming communication in C 30**](#_Toc106539517)

[**3.6.3 Simulation 32**](#_Toc106539518)

[**3.6.4 Program 8051 for serial data transfer 35**](#_Toc106539519)

[**3.6.5 Conclude on transmitting serial data. 36**](#_Toc106539520)

[**CONCLUSION 37**](#_Toc106539521)

[**REFERENCES 38**](#_Toc106539522)

# CHAPTER 1: INTRODUCTION

## 1.1 Introduction to Computer organization and architecture.

When the computer was born, it laid the foundation for a Digital Age and its applications in industrial automation are still in high demand today.

With the subject of computer architecture and organization, we have learned about the architecture of a computer including: central processing unit (CPU), arithmetic processing unit (ALU), memory peripheral unit (I/O), bus system. So how to apply computers to industrial control?.

To find that answer, the team delved into research and found that when a computer is miniaturized and integrated on one computer for application in controlling electronic devices or used in embedded systems, called microcontroller. Besides, to communicate with the external environment as well as the computer, the microcontroller also needs interrupt and communication operations.

In learning computer organization and architecture, we know that 8051 is one of the most popular microcontrollers. For understanding that useful abilities, we decided to deep into more. This period was characterized by innovation and diversity in computer architecture with the development of computing. This interaction, and it certainly was that, kick-started our own interest in computer architecture and organization, with particular emphasis on 8051 microcontrollers in computer architecture. Besides, we are using C, the high programming language for compiler the communication between computer and microcontroller.

## 1.2 Introduction to 8051 microcontrollers.

### **1.2.1 Brief History of 8051**

The first microprocessor 4004 was invented by Intel Corporation. 8085 and 8086 microprocessors were also invented by Intel. In 1981, Intel introduced an 8-bit microcontroller called the 8051. It was referred as system on a chip because it had 128 bytes of RAM, 4K byte of on-chip ROM, two timers, one serial port, and 4 ports (8-bit wide), all on a single chip. When it became widely popular, Intel allowed other manufacturers to make and market different flavors of 8051 with its code compatible with 8051. It means that if you write your program for one flavor of 8051, it will run on other flavors too, regardless of the manufacturer. This has led to several versions with different speeds and amounts of on-chip RAM.

### **1.2.2 Comparison between 8051 Family Members**

**Table I. Compares the features available in 8051, 8052, and 8031.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **8051** | **8052** | **8031** |
| ROM(bytes) | 4K | 8K | 0K |
| RAM(bytes) | 128 | 256 | 128 |
| Timers | 2 | 3 | 2 |
| I/O pins | 32 | 32 | 32 |
| Serial port | 1 | 1 | 1 |
| Interrupt sources | 6 | 8 | 6 |

**Picture about 8051 family member:**

A picture containing text, electronics, circuit

Description automatically generated

**Figure 1.1*:* AT89C51 (a member of 8051 family)**

Diagram, schematic

Description automatically generated

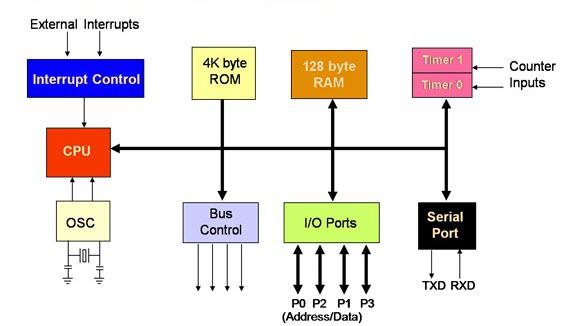
**Figure 1.2:Block diagram of AT89C51**

### **1.2.3 Features of 8051 Microcontroller**

An 8051 microcontroller comes bundled with the following features :

* 4KB bytes on-chip program memory (ROM)
* 128 bytes on-chip data memory (RAM)
* Four register banks
* 128 user defined software flags
* 8-bit bidirectional data bus
* 16-bit unidirectional address bus
* 32 general purpose registers each of 8-bit
* 16 bit Timers (usually 2, but may have more or less)
* Three internal and two external Interrupts
* Four 8-bit ports,(short model have two 8-bit ports)
* 16-bit program counter and data pointer
* 8051 may also have a number of special features such as UARTs, ADC, Op-amp, etc

### **1.2.4 Block Diagram and architechure of 8051 Microcontroller**



**Figure 1.3:** **The block diagram of an 8051 microcontroller**

Diagram, schematic

Description automatically generated

**Figure 1.4:** **Architecture of an 8051 microcontroller**

### **1.2.5 8051 Microcontroller Pin diagram**

8051 microcontroller is available in 40 pin Dual Inline package (DIP). All these 40 pins perform different functions like read, write, interrupts, I/O operations, address etc. 8051 microcontroller have four I/O ports (Port-1, Port-2, Port-3, Port-4 ) where in each port has 8 pin. Each pin configured as a input pin or output pin depends on logic state of the pin. Therefore, out of 40 pins, 32 pins are allotted for I/O port. The remaining pins are assigned to VCC, GND, XTAL1, XTAL2, RST, ALE, EA’ and PSEN’.

Now, the microcontroller 8051’s pin diagram and explanation is given below.

Diagram

Description automatically generated

**Figure 1.5:****The microcontroller 8051’s pin diagram**

Pin 1 to Pin 8 (Port – 1) – Pin 1 to Pin 8 is assigned to Port 1 for simple I/O operations. These ports are work as a bidirectional port. It means all the pins of port 1 work as a input pin or output pins. If Logic 1 (one) is applied to the I/O port it will act as a input pin and if logic 0 (zero) is applied to the I/O port it will act as a output pin.

Pin 9 (RST) – It is a reset input Pin, which is used to reset the 8051 microcontrollers to its initial values when logic 1 is applied to this pin. It is active high pin.

Pin 10 to Pin 17 (Port-3) – Pin 10 to Pin 17 are assigned to Port 3. This port is also a bidirectional I/O port like port 1. This port performs some special functions like interrupts, control signals, timer input, serial communication etc. The detail function of each pins are given below:

P10 (RXD) – Pin 10 is used as a RXD (serial data receive pin) which is for serial input pin. By using this input signal microcontroller receives data for serial communication.

P11 (TXD) – Pin 11 is used as a TXD (serial data transmit pin) which is serial output pin. By using this output signal microcontroller transmits data for serial communication.

P12 and P13 (INT0′, INT1′ ) – Pins12 and 13 are used for External Hardware Interrupt 0 and Interrupt 1 respectively. When this interrupt is activated (i.e. when it is low), 8051 gets interrupted means it stopped whatever it is doing and jumps to the vector table where ISR’s (Interrupt Service Routine) are stored and starts performing Interrupt Service Routine (ISR) from that vector location.

P14 and P15 (T0 and T1) – Pin 14 and 15 are used for Timer 0 and Timer 1 external input. They can be connected with 16-bit timer/counter.

P16 (WR’) – Pin 16 is used for external memory write i.e. writing data to the external memory.

P17 (RD’) – Pin 17 is used for external memory read i.e. reading data from external memory.

Pin 18 and Pin 19 (XTAL2 And XTAL1) –Pins 18 and 19 i.e. XTAL 2 and XTAL 1 are the pins for interfacing external oscillator. Mostly, a Quartz Crystal Oscillator is connected here to get the system clock.

# CHAPTER 2: PROGRAMMING INTERRUPT THE 8051 MICROCONTROLLERS USING THE C PROGRAMMING LANGUAGE.

## 2.1 Interrupt in 8051

At any given instant, multiple processes can be running in the 8051. Additionally, a number of external devices can also be connected to it at the same time. So it's safe to say that multiple peripherals or processes could require the CPU at any given moment. As the 8051 has a single-core CPU it can tend to only one task at a given instant of time. So Interrupt here is a solution for two or more resources needing CPU at the same time. An interrupt is a signal sent by hardware or software processes calling for the immediate attention of the CPU. Once the CPU receives this signal it stops whatever it is doing and takes care of that particular resource.

## 2.2 Types of interrupts in 8051

**Diagram

Description automatically generated**

**Figure 2.1**: **Interrupt of 8051 Microcontroller**

In fact, there are only 5 interrupts for users in 8051 but manufacturers say there are 6 interrupts because they count the reset command. The six interrupts of 8051 are distributed as follows:

**Reset**: When the pin reset is activated from 8051, the dance program counter to the address 0000H. This is the address to turn on the source.

Two interrupts for timer sets: 1 for **Timer0** and 1 for **Timer1**. The corresponding address of these interrupts is 000BH and 001BH.

Two interrupts for external hardware interrupts: Pins 12 (P3.2) and 13 (P3.3) of P3 ports are external hardware interrupts **INT0** and **INT1** respectively. The corresponding address of these external interrupts is 0003H and 0013H.

**Serial communication**: There is 1 general interrupt for both receiving and transmitting serial data. The address of this interrupt in the interrupt vector table is 0023H.

## 2.3 Interrupt priorities

Whenever there is any device that needs to be served, it notify the microcontroller by sending an interrupt signal. Upon receiving the interrupt signal, the microcontroller stops all that it is doing to switch to the calling device. The interrupt program is called the ISR (Interrupt Service Routine), also known as Interrupt Handler. After the service is finished, the processor returns to the previous interrupt point and continues to perform the job.

For each interrupt, there must be a interrupt service (ISR) or a interrupt management process to give the microcontroller's task when called interrupted. When an interrupt is called, the microcontroller will run the service. For each interrupt, there is a fixed position in memory to keep its ISR address. The memory position group is dedicated to storing the addresses of the ISRs called the interrupt vector table.

**Table II. The interrupt vector table of 8051.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Interrupt** | **Interrupt flag** | **Interrupt server address** | **Interrupt sequence number** |
| **Reset** | **-** | **0000h** | **-** |
| **External interrupt 0** | **IE0** | **0003h** | **0** |
| **Timer 0** | **TF0** | **000Bh** | **1** |
| **External interrupt 1** | **IE1** | **0013h** | **2** |
| **Timer 1** | **TF1** | **001Bh** | **3** |
| **Interrupt communication** | **RI/TI** | **0023h** | **4** |

## 2.4 The process when performing an interrupt

When activating a microcontroller, perform the following steps:

**Step 1:** It completes the command that is executed and saved the address of the next command to the stack.

**Step 2:** It also saves the current situation of all interrupts.

**Step 3:** It jumps to a fixed position in memory called the interrupt vector table, which stores the address of an interrupt service.

**Step 4:** The microcontroller receives the ISR address from the interrupt vector table and jumps there. It starts to perform the interrupt service to the final command of ISR and returns to the main program from interrupt.

**Step 5**: When the microcontroller returns to where it is interrupted. First, it receives the address of the PC program counter from the stack by dragging 02 bytes on the top of the PC stack. Then start executing commands from that address.

## 2.5 Interrupt stucture of 8051 Microcontroller

Upon ‘RESET’ all the interrupts get disabled, and therefore, all these interrupts must be enabled by a software. In all these five interrupts, if anyone or all are activated, this sets the corresponding interrupt flags as shown in the figure. All these interrupts can be set or cleared by bit in some special function register that is Interrupt Enabled (IE), and this in turn depends on the priority, which is executed by IP interrupt priority register.

Diagram, engineering drawing, schematic

Description automatically generated

**Figure 2.2: Interrupt structure of 8051 Microcontroller**

Interrupt Enable (IE) Register: This register is responsible for enabling and disabling the interrupt. It is a bit addressable register in which EA must be set to one for enabling interrupts. The corresponding bit in this register enables particular interrupt like timer, external and serial inputs. In the below IE register, bit corresponding to 1 activates the interrupt and 0 disables the interrupt.

**Table III. The control register of interrupt ( Enable interrupt)**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| IE |  | EA | - | - | ES | ET1 | EX1 | ET0 | EX0 |

**IE.7 EA** Disables all interrupt. If EA=0, no interrupt will be acknowledged. If EA=1, interrupt source is individually enable or disabled by setting or clearing it enable bits.

**IE.6 -** Not implemented, reserved for future use\*.

**IE.5 -** Not implemented, reserved for future use\*.

**IE.4 ES** Enable or disable the Serial port interrupt.

**IE.3 ET1** Enable or disable the Timer 1 overflow interrupt.

**IE.2 EX1** Enable or disable External interrupt 1.

**IE.1 ET0** Enable or disable the Timer 0 overflow interrupt.

**IE.0 EX0** Enable or disable External interrupt 0.

When the source is turned on, all interrupts are banned (covered), which means that no interrupts are responded by the microcontroller unless they are activated. The interrupts must be activated by the software for the microcontroller to respond. There is a register called an IE (Interrupt Enable) spot - at A8H address responsible for permission and banning interrupts.

In order to allow an interrupt, we must take the following steps:

**Step 1:** If EA = 0, no interrupt is met even though its corresponding bit in IE is of high value. The D7 - EA bit of IE register must be turned on to allow the remaining bits of the operating register.

**Step 2:** If EA = 1 then all interrupts are allowed and will be met if their corresponding bits in IE are high.

**Table IV. The TCON. The counter control register**

Table

Description automatically generated

## 2.6 Interrupt Timer

### **2.6.1 Block diagram**

Diagram

Description automatically generated

**Figure 2.3: Block diagram interrupts timer**

### **2.6.2 Application**

Using the Timer 0 interrupt to control 8 LEDs flashing at port P2 and timer every 10 seconds to turn on the LED on Port P1\_3, the lights will turn off after a few second.( the frequency of 8051 is XTAL=12MHz)

### **2.6.3 Programming interrupt in C**

#include <REGX51.H>

void delay (int time){

while(time--);

}

int t=0;

void main(){

while (1){

TMOD=0x01;//timer 0, mode 1

ET0=1;// Configure interrupt 0 for falling edge on

INT0;

TR0=1;

EA=1;//Enable Global Interrupt flag

P2=0;

delay(15000);

P2=0xff;

delay(15000);

}

}

void ngat() interrupt 1{

long a=50000;

t++;// dem thoi gian

TH0=0xfc;

TL0=0x18;

TR0=1;// khoi dong bo dinh thoi

if (t>=10000){ // t dat truoc thoi gian 10s

t=0;

P1\_3=0;

while(a--){};

P1\_3 =1;

}

}

### **2.6.4 Circuit diagram.**

Diagram, schematic

Description automatically generated

**Figure 2.4: Circuit diagram interrupts timer.**

### **2.6.5 Simulation**

Intially, there are only 8 LEDs flashing at port P2.

Schematic

Description automatically generated

**Figure 2.5: The LED turn off/on continuously.**

After 10 second, LED on Port P1\_3 turn on.

Chart, schematic

Description automatically generated

**Figure 2.6: The LED on port P1\_3 turn on after 10 second.**

## 2.7 Interrupt external

In the AT89C52, there are two external interrupt inputs at pins INT0 (P3.2) and INT1 (P3.3). When there is an event on the INT pin (edge down or level 0) will generate an interrupt (if enabled).

**2.7.1 Block diagram**

**Diagram

Description automatically generated**

**Figure 2.7: Block diagram interrupts external**

### **2.7.2 Application**

Using the External interrupt 0 to control 2 Led.

### **2.7.3** **Programming interrupt in C**

#include<main.h>

sbit LED1 = P1^0;

sbit LED2 = P1^1;

int main()

{

EA = 1;

EX0 = 1;

IT0 = 1;

LED2=0;

while(1)

{

LED1=1;

delay\_ms(200);

LED1=0;

delay\_ms(200);

}

}

void External0\_ISR() interrupt 0

{

LED2 = ~LED2;

}

### **2.7.4 Circuit diagram.**

Chart

Description automatically generated

**Figure 2.8: Circuit diagram interrupts external.**

### **2.7.5 Simulation**

Initially, LED D1 flashing.

A screenshot of a computer

Description automatically generated with low confidence

**Figure 2.9: The LED D1 turn off/on continuously.**

After that, you press down button at Port 3, LED D2 will turn on.

Chart

Description automatically generated

**Figure 2.10: The LED D2 turn on after press down button.**

# CHAPTER 3: PROGRAMMING UART THE 8051 MICROCONTROLLERS USING THE C PROGRAMMING LANGUAGE.

## 3.1 Introduction to UART

One of the 8051’s many powerful features is it’s integrated Universal asynchronous receiver/transmitters (UART), otherwise known as a serial communication. The fact that the 8051 has an integrated serial port means that you may very easily read and write values to the serial port. If it were not for the integrated serial port, writing a byte to a serial line would be a rather tedious process requiring turning on and off one of the I/O lines in rapid succession to properly “clock out” each individual bit, including start bits, stop bits and parity bits.

However, we do not have to do this. Instead, we simply need to configure the serial port’s operation mode and baud rate. Once configured, all we have to do is write to an SFR to write a value to the serial port or read the same SFR to read a value from the serial port. The 8051 will automatically let us know when it has finished sending the character we wrote and will also let us know whenever it has received a byte so that we can process it. We do not have to worry about transmission at the bit level-which saves us quite a bit of coding and processing time.

### **3.1.1 Registers used for UART**

- SCON (Serial Control Register) – Bit Addressable

- SBUF (Serial Buffer Register) – Byte Addressable

- PCON (Power Control Register) – Byte Addressable

**SCON (Serial Control Register)**

This register is bit addressable.

**Table V.** **8 address bits in SCON**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| SM0 | SM1 | SM2 | REN | TB8 | RB8 | TI | RI |

**SM0:** Serial Port Mode Specifier 1 bit.

**SM1:** Serial port Mode Specifier 2 bit

These Two bits are used to select the Mode of the UART.

**Table VI. Function of SM0 and SM1 in Different Mode**

|  |  |  |  |
| --- | --- | --- | --- |
| SM0 | SM1 | Mode | Baud rate |
| 0 | 0 | Shift Register (Mode 0) | Fosc/12 |
| 0 | 1 | 8-bit UART (Mode 1) | Variable (Set by Timer 1) |
| 1 | 0 | 9-bit UART (Mode 2) | Fosc/32 or Fosc/64 |
| 1 | 1 | 9-bit UART (Mode 3) | Variable (Set by Timer 1) |

**SM2:** Multiprocessor communications bit. Set/cleared by program to enable multiprocessor communications in modes 2 and 3. When set to 1 an interrupt is generated if bit 9 of the received data is a 1; no interrupt is generated if bit 9 is a 0. If set to 1 for mode 1, no interrupt will be generated unless a valid stop bit is received. Clear to 0 if mode 0 is in use.

**REN:** Receive enable bit. Set to 1 to enable reception; cleared to 0 to disable reception.

**TB8:** Transmitted bit 8. Set/cleared by program in modes 2 and 3.

**RB8:** Received bit 8. Bit 8 of received data in modes 2 and 3; stop bit in mode1. Not used in mode 0.

**TI:** Transmit Interrupt flag. Set to one at the end of bit 7 time in mode 0, and at the beginning of the stop bit for other modes. Must be cleared by the program.

**RI:**  Receive Interrupt flag. Set to one at the end of bit 7 time in mode 0, and halfway through the stop bit for other moves. Must be cleared by the program.

**SBUF (Serial Buffer Register)**

SBUF Register: For a byte of data to be transferred via the TxD line, it must be placed in the SBUF. SBUF holds the byte of data when it is received by the MCS51’s RxD line.

Diagram

Description automatically generated

**Figure 3.1:** **Block diagram of** **Serial buffer register**

**PCON (Power Control Register)**

This Register is not Bit Addressable.

**Table VII.** **8 address bits in PCON**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| SMOD | - | - | - | GF1 |  |  | GF0 |

**SMOD:** Double baud rate bit. If Timer 1 is used to generate baud rate and SMOD = 1, the baud rate is doubled when the serial port is used in modes 1, 2, or 3.

**GF1:** General-purpose flag bit.

**GF0:**  General-purpose flag bit.

**PD:** Power Down bit. Setting this bit activates the Power Down operation in the 8051BH. (Available only in CHMOS).

**IDL:** Idle Mode bit. Setting this bit activates Idle Mode operation in the 8051BH. (Available only in CHMOS).

### **3.1.2 Baud rate in the 8051**

The 8051 transfers and receives data serially at many different baud rates. The baud rate in the 8051 is programmable. This is done with the help of Timer 1. The relationship between the crystal frequency and the baud rate in the 8051 is shown below. The 8051 divides the crystal frequency by 12 to get the machine cycle frequency.

In the case XTAL = 11.0592 MHz, the machine cycle frequency is 921.6 KHz (11.0592MHz/12 = 921.6 KHz). The 8051’s serial communication UART circuitry divides the machine cycle frequency of 921.6kHz by 32 once more before it is used by Timer 1 to set the baud rate. Therefore, 921.6 kHz divided by 32 gives 28,800 Hz.

## 3.2 Serial – Parallel Transfer

***Serial Transmission:***

In Serial Transmission, data-bit flows from one computer to another computer in bi-direction. In this transmission, one bit flows at one clock pulse. In Serial Transmission, 8 bits are transferred at a time having a start and stop bit.

Diagram

Description automatically generated

**Figure 3.2:** **Serial Transfer from Sender to Receiver**

***Parallel transmission:***

In Parallel Transmission, many bits are flow together simultaneously from one computer to another computer. Parallel Transmission is faster than serial transmission to transmit the bits. Parallel transmission is used for short distance.

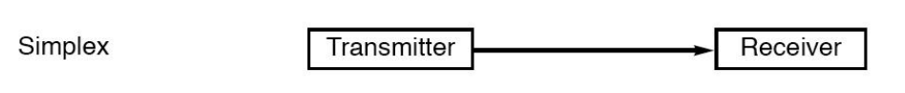
**Diagram

Description automatically generatedFigure 3.3:** **Parallel** **Transfer from Sender to Receiver**

## 3.3 Types of Data Transmission

### **a) Simplex mode**

In Simplex mode, the communication is unidirectional, as on a one-way street. Only one of the two devices on a link can transmit, the other can only receive. The simplex mode can use the entire capacity of the channel to send data in one direction.



**Figure 3.4:** **Simplex mode**

### **b) Half-duplex mode**

In half-duplex mode, each station can both transmit and receive, but not at the same time. When one device is sending, the other can only receive, and vice versa. The half-duplex mode is used in cases where there is no need for communication in both directions at the same time. The entire capacity of the channel can be utilized for each direction.

Diagram

Description automatically generated

**Figure 3.5: Half-duplex mode**

### **c) Full-duplex mode**

In full-duplex mode, both stations can transmit and receive simultaneously. In full-duplex mode, signals going in one direction share the capacity of the link with signals going in another direction, this sharing can occur in two ways:

Either the link must contain two physically separate transmission paths, one for sending and the other for receiving. Or the capacity is divided between signals traveling in both directions.

Full-duplex mode is used when communication in both directions is required all the Diagram

Description automatically generatedtime. The capacity of the channel, however, must be divided between the two directions.

**Figure 3.6:** **Full duplex mode**

### **d) Transmission mode in 8051**

**Mode 0:**

The TXD (which originally stands for transmission) pin (Port 3.1) is used to send/receive the synchronizing clock signal.

The RXD (which stands for Receive) pin (Port 3.1) is used for transferring and receiving the data. Reception of data requires REN=1 and RI=0.

In this mode, data is transferred as it would in a shift register—bit by bit with a common synchronizing clock signal.

Diagram

Description automatically generated

**Figure 3.7: Mode 0**

**Mode 1:**

Mode 1 is the most commonly used mode for UART based serial communication.

Here the 8051 follows a full-duplex mode of communication and can transmit and receive data at the same time.

Diagram

Description automatically generated

**Figure 3.8:** **Mode 1**

**Mode 2:**

This mode also implements UART communication but can be used to transfer/receive 9 bits of data.

TB8/RB8 store the 9th bit during the same.

For mode 2 in full-duplex, the value 90H should be inserted into the SCON register.

For mode 2 in half-duplex, the value 80H should be inserted into the SCON register.

**Mode 3:**

This mode is similar to mode 2 but can transfer data at variable baud rates generated by timer 1.

For mode 2 in full-duplex, the value 130H should be inserted into the SCON register.

For mode 2 in half-duplex, the value 120H should be inserted into the SCON register.

## 3.4 Method of serial communication

Diagram

Description automatically generatedSerial data communication uses two methods, ***synchronous*** and ***asynchronous***. The synchronous method transfers a bulk data in the framed structure at a time, while the asynchronous method transfers a single byte data in the framed structure at a time. Besides, the method in 8051’s microcontroller is asynchronous communication.

**Figure 3.9:** **Asynchronous communication**

In ***asynchronous*** the data coming in at the receiving end of the data line in a serial data transfer is all 0s and 1s, it is difficult to make sense of the data unless the sender and receiver agree on a set of rules, a protocol, on how the data is packed, how many bits constitute a character, and when the data begins and ends.

***Asynchronous*** serial data communication is widely used for character-oriented transmissions. In this method, each character is placed between start and stop bits. This is called framing.

In data framing for ***asynchronous*** communications, the data such as ASCII characters, are packed between a start bit and stop bit. The start bit is always one bit, but the stop bit can be one or two bits. The start bit is always a 0 (low) and the stop bit is 1(high). Look at below figure in which the ASCII character “A” (8-bit binary 0100 0001) is framed between the start bit and a stop bit. Notice that the LSB is sent out first.

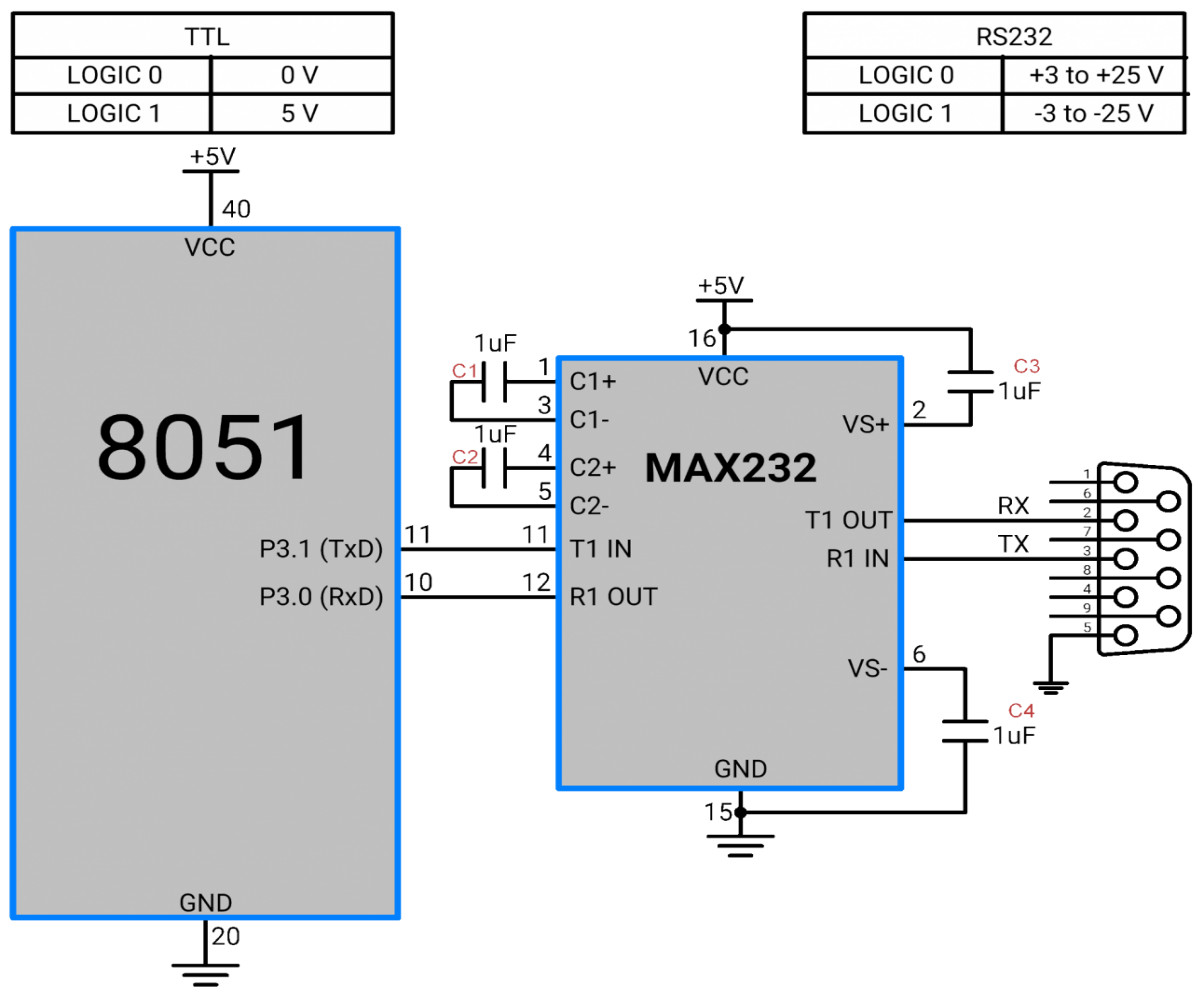
## 3.5 Interface standard.

### **Diagram Description automatically generateda) RS 232**

**Figure 3.10:** **RS232’s port**

RS 232 is an interfacing standard and is the most widely used serial I/O interfacing standard. Since the standard was set long before the advent of the TTL logic family, its input and output voltage levels are not TTL compatible. For this reason, to connect any RS232 to a microcontroller system we must use voltage converters such as MAX232 to convert the TTL logic levels to RS232 voltage levels, and vice versa. MAX232 IC’s are commonly referred to as line drivers.

### **b) Max232**



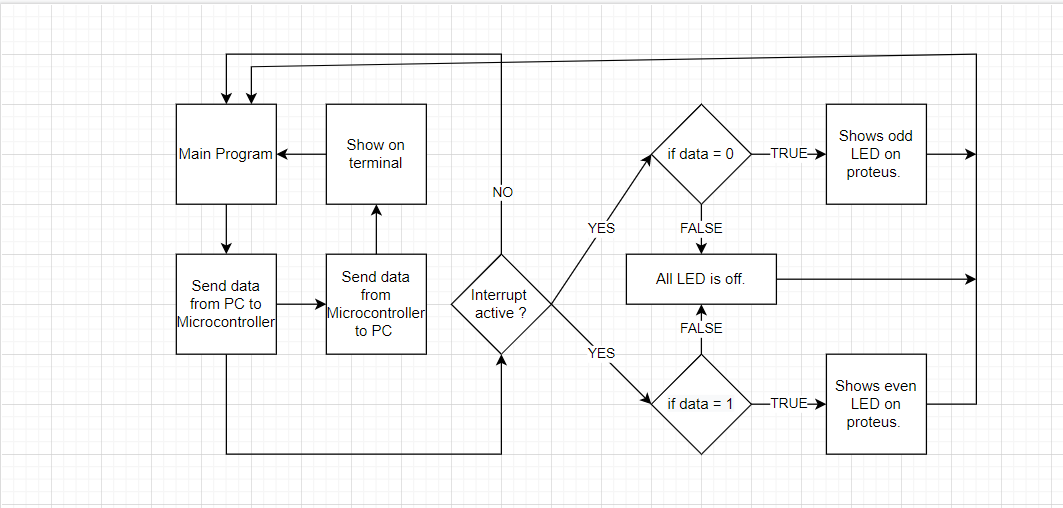
**Figure 3.11:** **Using MAX232 connected between 8051 and RS232**

Since the RS232 is not compatible with today’s microprocessors and microcontrollers, we need a voltage converter (line driver) to convert the RS232’s signals to TTL voltage levels that will be acceptable to the 8051’s TxD and RxD pins. One example of such converter is MAX232. It converts from RS232 voltage levels to TTL voltage levels and vice versa.

One advantage of the MAX232 is that it uses a +5 V power source which is the same as the source voltage for the 8051. The MAX232 has two sets of line drivers for transferring and receiving data. The line drivers used for TxD are called T1 and T2, while the line drivers for RxD are designated as R1 and R2. MAX232 requires four capacitors ranging from 1 to 22 µF. The most widely used value for these capacitors is 22 µF.

## 3.6 UART – Universal Asynchronous Receiver/Transmitter

### **3.6.1 Block diagram**



**Figure 3.12:** **Block diagram UART transmitting and interrupts**

### **3.6.2 Programming communication in C**

#include <at89x52.h>

#include <string.h>

#include <stdio.h>

// initial global variables

int I;

unsigned char rdata;

void sendchar(unsigned char a)

{

SBUF = a;

while(TI==0){}

TI=0;

}

void sendstring(char \*a)

{

int I,n;

n = strlen(a);

for(i=0;i<n;i++)

{

sendchar(a[i]);

}

}

void setup (){

TMOD = 0x20; //Select Timer1, Mode 2

SCON = 0x50; //Select Mode 1

TH1= 0xFD; //Set baudrate 9600

TR1 = 1; //Enable Timer1

IE = 0x90; //Enabling Serial Interrupt

}

void serint(void) interrupt 4 using 1 {

if(P2) // The loop checks the signal from the P2 pin.

{

rdata = SBUF; // Assign the data received to variable rdata

RI = 0; // Set RI flag

}

switch(rdata){

case(‘0’): {

// Set Odd LED

P2\_0 = 0; P2\_2 = 0; P2\_4 = 0; P2\_6 = 0;

P2\_1 = 1; P2\_3 = 1; P2\_5 = 1; P2\_7 = 1;

break;

}

case(‘1’): {

// Set Even LED

P2\_0 = 1; P2\_2 = 1; P2\_4 = 1; P2\_6 = 1;

P2\_1 = 0; P2\_3 = 0; P2\_5 = 0; P2\_7 = 0;

break;

}

default: {

// Turn off all LED

P2\_0 = 1; P2\_2 = 1; P2\_4 = 1; P2\_6 = 1;

P2\_1 = 1; P2\_3 = 1; P2\_5 = 1; P2\_7 = 1;

break;

}

}

}

void main ()

{

setup();

while(1){

sendstring(“BAUDRATE 9600 8051 MICROCONTROLLER FINAL PROJECT \n\r”);

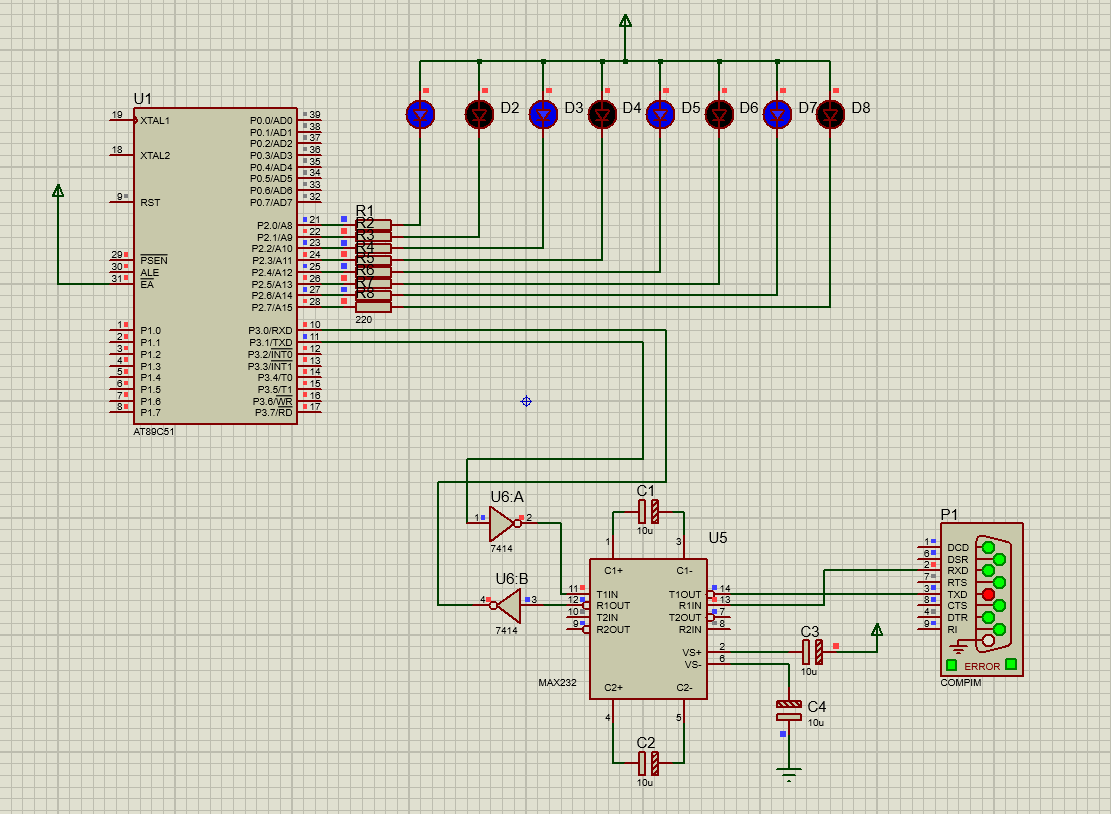
}

}

### **3.6.3 Simulation**

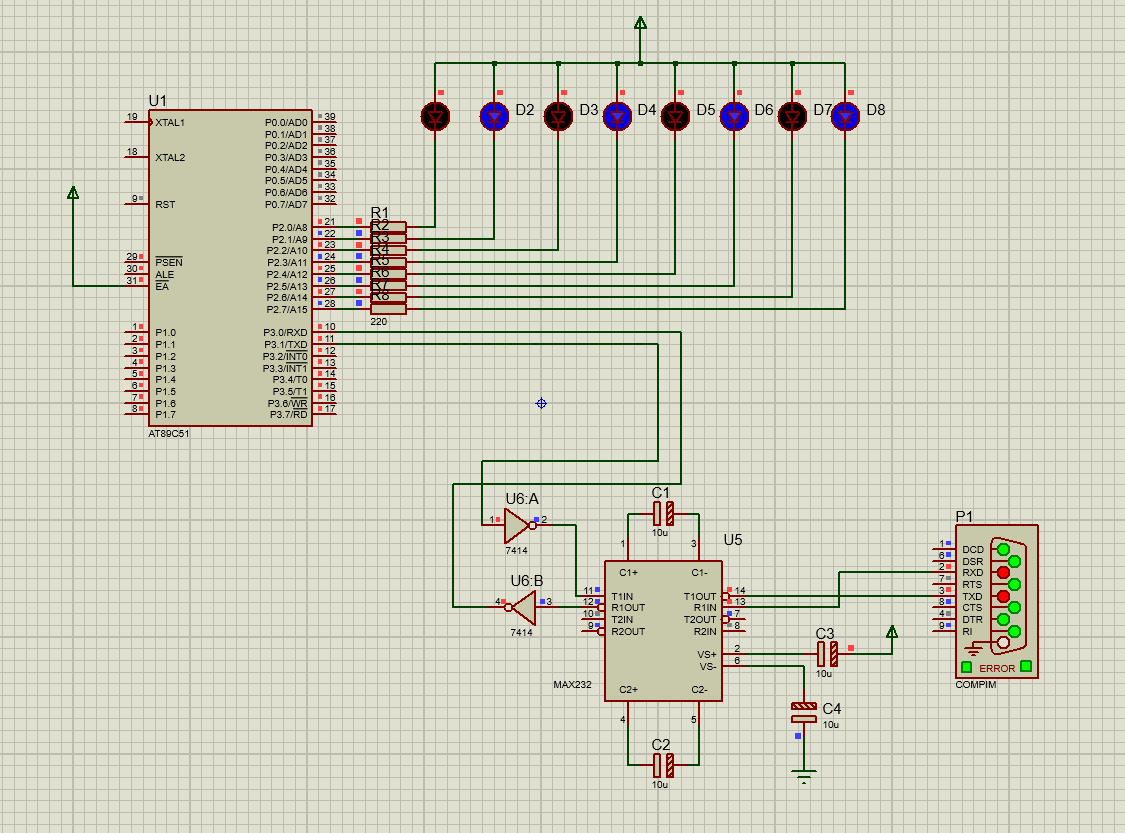
**Program 8051 to receive serial data.**

When we start simulating, we input 0 from the terminal. The RI receiving flag bit is checked with a loop to ensure the entire character has received enough (when RI=1). When the RI flag was established, there was a byte in the SBUF. Its contents need to be read immediately to avoid loss ( turn on ODD LED ). Then, to receive the following data, the processor will delete the RI: RI=0 receiving the flag.



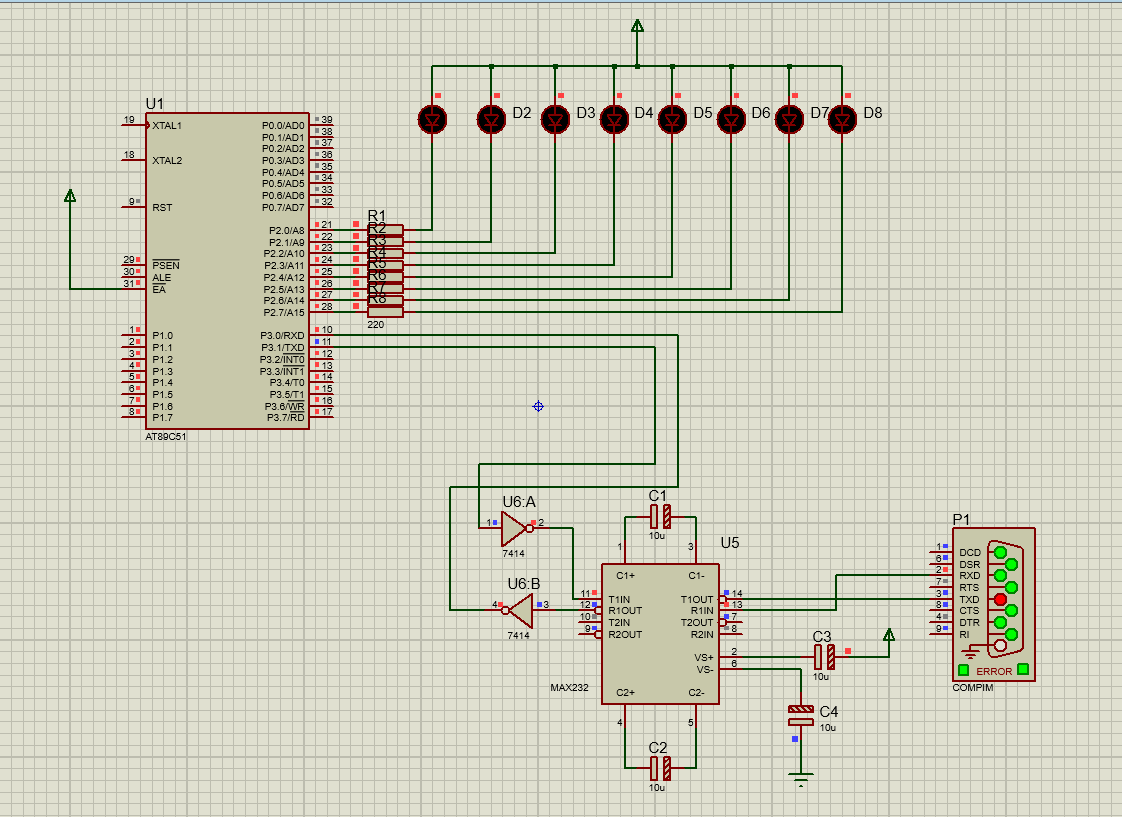
**Figure 3.13:** **Turn on ODD LED when input 0 from PC to Microcontroller**

When we input 1 from the terminal. The RI receiving flag bit is checked with a loop to ensure the entire character has received enough (when RI=1). When the RI flag was established, there was a byte in the SBUF. Its contents need to be read immediately to avoid loss (turn on EVEN LED ). Then, to receive the following data, the processor will delete the RI: RI=0 receiving the flag.



**Figure 3.14:** **Turn on EVEN LED when input 1 from PC to Microcontroller**

When we input anything from the terminal. The program will turn off all LEDs.



**Figure 3.15:** **Turn off all LED when input anything from PC to Microcontroller**

**Conclude on receiving serial data.**

When receiving bits through its RxD pin, 8051 must go through the following steps:

- It receives the Start bit which indicates that the bit after it is the first bit of data to receive.

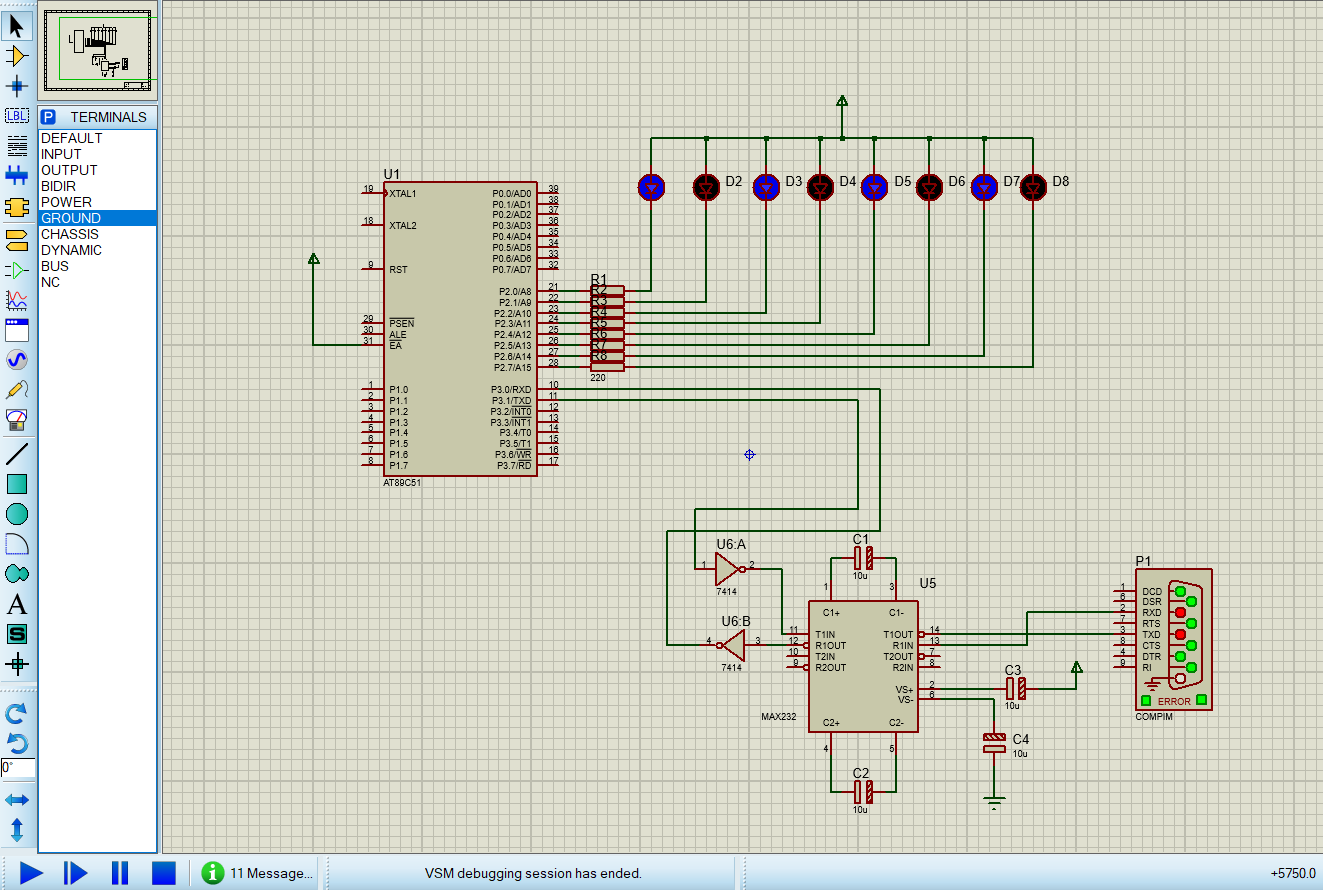
- 8-bit characters are received one bit at a time. When the last bit is received, a byte is formed and placed in the SBUF.

- When the Stop bit is received, 8051 turns on RT = 1 to indicate that the entire character is received and must be removed before it is received byte to be overwritten.

- By checking the RI flag bit when it is turned on we know that a character has been received and is in the SBUF. Copy SBUF content to a safe place in another register or memory bar before it's lost.

- After the SBUF is recorded in a safe place, the RI flag is removed to 0 to prepare for the next cycle check.

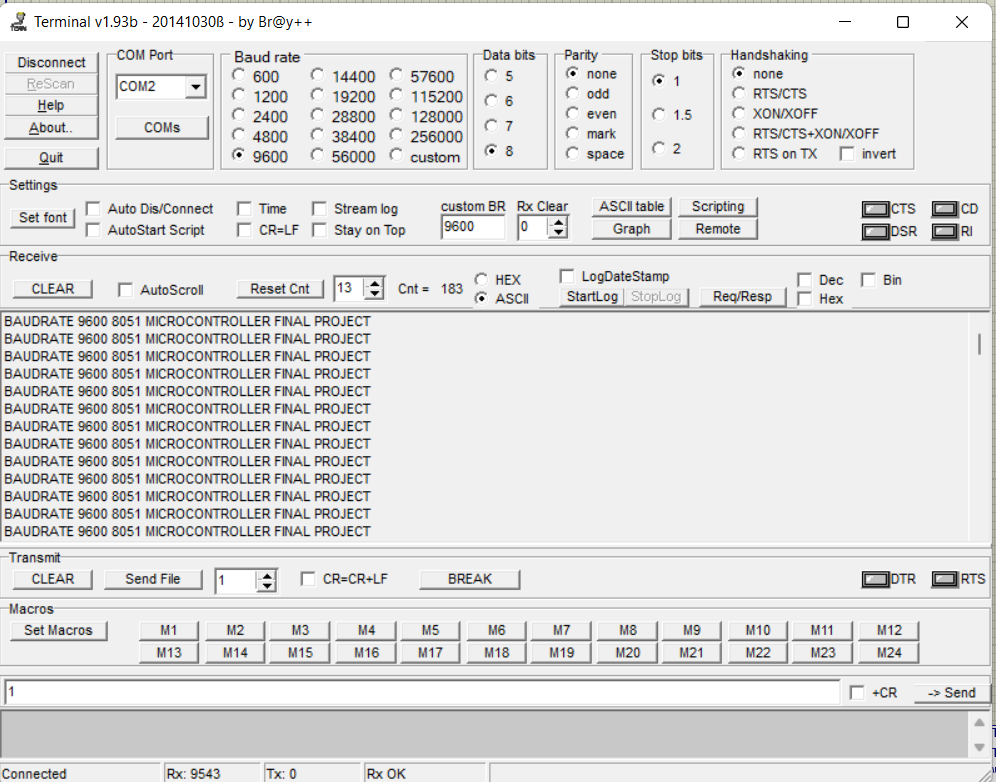
When the simulation is turned off, we still retain the state before turning it off.



**Figure 3.16:** **Keep the latest state before turning it off.**

### **3.6.4 Program 8051 for serial data transfer**

Each character that needs to be transmitted is written to the SBUF. After the first START bit followed by the data byte and at last STOP bit forms a 10-bit frame. During the Stop bit transfer, the TI flag is turned on (TI = 1) by 8051 to be ready to transmit the next character. Before the SBUF is loaded with a new byte, the TI flag must be removed to check for the next data transfer.



**Figure 3.17:** **The data from PC to Microcontroller will show in the terminal**

### **3.6.5 Conclude on transmitting serial data.**

To understand the importance of the TI break flag let's look at the sequence of steps below that 8051 must take when transmitting a character via TxD:

- Bytes characters that need to be transmitted are written to SBUF.

- Start bit transfer

- Transfer 8-bit characters one by one.

- Bit Stop is transmitted, during the stop bit transmission, the TI flag is turned on (TI = 1) by 8051 to be ready to transmit the next character.

- By checking the TI flag, we know for sure that we do not load too quickly into the SBUF register. If we load a byte into the SBUF before the TV recording is turned on, the data of the previously untranslated byte will be lost. We have to wait for 8051 to turn on the TV flag so that the newspaper has passed one byte and it is ready to transmit the next byte.

- Before the SBUF is loaded with a new byte, the TI flag must be removed for inspection for the next data transfer.

# CONCLUSION

In this topic, the group studied and focused on 8051 microcontrollers using C programming language for interrupts and communication. The group has gone deep into learning and implementing a number of small applications such as controlling 8 single leds, using push buttons, communicating from the computer's Com port via IC Max232 to the 8051 microcontroller. However, the team has not yet gone deep and exploited all the applications of the 8051 microcontroller.

From the results obtained, if we have more time and knowledge about sensors, peripherals and microcontroller programming, we can orient to expand the topic and apply it to reality such as traffic lights, perpetual calendar, LED clock, LCD temperature sensor, .....

# REFERENCES

<https://www.tutorialspoint.com/embedded_systems/es_microcontroller.htm> [Date 5-6-2022]

<https://www.academia.edu/6159308/The_8051_Microcontroller> [Date 5-6-2022]

https://quick-learn.in/8051-microcontroller-pin-diagram-and-pin-discription/ [Date 5-6-2022]

<https://exploreembedded.com/wiki/A4.8051_Communication_Protocols:UART,_RS232> [Date 15 – 05 - 2022]

<https://www.geeksforgeeks.org/difference-between-simplex-half-duplex-and-full-duplex-transmission-modes/> [Date 16 – 05 - 2022]

<https://www.sciencedirect.com/topics/computer-science/serial-communication> [Date 17 – 05 - 2022]

<https://technobyte.org/8051-uart-serial-communication-working/> [Date 17 – 05 - 2022]

Thomas W. Schultz - C and the 8051 Hardware, Modular Programming & Multitasking (1997, Prentice Hall PTR) - libgen.lc

THỰC HÀNH KIẾN TRÚC VÀ TỔ CHỨC MÁY TÍNH (Sách tham khảo dành cho sinh viên Ngành Công nghệ Kỹ thuật Máy Tính) – PhD. Phạm Văn Khoa

Vi Xử Lý 1 – Nguyễn Đình Phú – Năm 2006

8051 Tutorial (C) Copyright 1997, 1998, 1999 by Vault Information Services. All Rights Reserved.

Giáo trình Vi xử lý – Vi điều khiển – Trường Đại học Sao Đỏ - Bộ Công thương

Microcontroller Projects in C for the 8051 – Dogan Ibrahim 2000